TrafficLLM: Enhancing Large Language Models for Network Traffic Analysis with Robust Traffic Representation

Tianyu Cui[1], Xinjie Lin[1], Sijia Li[1], Qilei Yin[1], Qi Li[1,2], Ke Xu[1,2]
[1] *Zhongguancun Laboratory,* [2] *Tsinghua University*

Abstract

Network traffic analysis enables many machine learning (ML) based models for the detection of threats and maintenance of service quality, but encountering difficulty in generalization across different tasks and unseen data. Large language models (LLMs), as a promising avenue to produce the general solution across various tasks, have successes in many specific fields with strong generalizations. However, their assistance for traffic analysis is insufficient due to the limitations of adapting to traffic domains. In this paper, we propose TrafficLLM, a universal LLM adaptation framework for network traffic analysis. TrafficLLM introduces a general solution to answer how to achieve robust traffic representation through generalization on heterogeneous traffic data, multimodal learning across different tasks, and new environments of network traffic. Through moderate learning, TrafficLLM can release two core capabilities, i.e., traffic detection and generation, on a wide range of downstream tasks, like encrypted traffic classification and APT detection. TrafficLLM can reach F1-scores of 0.9875 and 0.9483 with at most 80.12% and 33.92% better performances compared to 15 state-of-the-art solutions on traffic detection and generation respectively. We further showcase TrafficLLM's strong generalization on unseen data and real-world settings.

1 Introduction

Network traffic is the cornerstone of the Internet, carrying all interactions and data transfers within the network. However, as networking techniques evolve, attackers can leverage network traffic as a medium to conduct various malicious activities, such as fishing [30], malware campaigns [24], web attacks [38], and exploiting vulnerabilities [56]. Considerable enterprises have recognized the importance of analyzing traffic data to detect threats, investigate incidents, and monitor environments [10, 63]. This has facilitated the development of many sophisticated network traffic analyzers (NTA) and security information and event management (SIEM) solutions, e.g., Cisco Secure Network Analytics [67] and Rapid7 InsightIDR [58]. Existing work has achieved great improvement in many tasks, such as encrypted application classification [8, 52, 75, 89], website fingerprinting [17, 18, 28, 79], and malicious traffic detection [22–24, 57].

In recent years, machine learning (ML) based methods [2, 28, 39, 41, 41, 44, 55, 62, 65, 69, 75] have been proposed due to their strong representation-learning abilities for diverse traffic patterns. Despite their promising potential, ML-based methods still suffer from the following limitations, leading to low generalization of existing ML-based models: *(i) Generalization across various tasks*. In each sub-field of traffic analysis tasks, existing methods usually learn with handcrafted features and supervised labels to develop complicated ML models for specific tasks [40, 46]. These task-specific models are hardly shared across different tasks due to the specialized handcrafted features or model designs. The development costs will be considerable in covering various tasks. *(ii) Generalization to unseen data*. ML-based methods are widely criticized for their inability to handle unseen data [23]. These models are usually forced to learn known patterns in the high-quality labeled datasets. When faced with unseen data scenarios like concept drift [35, 85] and zero-day attacks [38, 68], ML models often achieve poor performance due to low generalization.

As a consequence, developing a more general model has been recognized as significant to enhancing generalization across different tasks and data distributions [27, 40]. Recently, large language models (LLMs) [9, 53, 73, 74, 87] have shown outstanding performance in many complex tasks [81, 83]. Thanks to their pattern mining, generalization to unseen data, and reproducibility across different tasks, LLMs could release remarkable capabilities in various downstream tasks [3, 15, 71, 90], which inspired multiple high-level views to develop specialized large-scale models for network traffic analysis. For instance, LLMs' pattern mining and reasoning ability can be utilized to learn robust representations behind IP attributes, flags, timings, and datagram lengths in traffic data. Moreover, the generalization ability allows LLMs to adapt to diverse network environments and attack scenarios. Therefore, LLM could serve as a more powerful ML model to provide robust

traffic representation with strong generalization ability.

However, the characteristics of network traffic analysis leave many challenges in using LLM for generalized traffic representation learning. First, the traffic data contains considerable heterogeneous features for pattern learning (e.g., protocol fields and statistical features), which is significantly different from the natural language. This input modality discrepancy from the plain text makes it difficult for native LLMs to process the traffic data [53, 74], which further prevents LLMs from generalizing to the different input formats of traffic features. Second, different downstream tasks involve diverse domain knowledge and traffic patterns (e.g., botnet traffic and Tor network traffic). Jointly learning the multi-type task-specific instruction semantics and traffic data can confuse LLM, leading to poor generalization across different tasks [27, 88]. Third, the traffic domain often faces environment drift like application updates and attack changes [38, 50, 85]. Unfortunately, LLM adaptation is extremely time-consuming due to the large model size. The high costs to update models for new environment generalization make LLM impractical in real-world scenarios.

To overcome these limitations, we propose TrafficLLM, a universal LLM adaptation framework to learn robust traffic representation for all open-sourced LLM in real-world scenarios and enhance the generalization across diverse traffic analysis tasks. TrafficLLM mainly implements three core designs: *(i) Traffic-domain tokenization*: To mitigate the input modality gap between traffic and language, TrafficLLM is equipped with a traffic-domain tokenization mechanism to extend LLM's native tokenizers. It helps LLM generalize to different types of traffic data and can achieve 106% efficiency improvement by reducing token length. *(ii) Dual-stage tuning pipeline*: TrafficLLM implements a dual-stage tuning pipeline to conduct multimodal learning with text and traffic data. This pipeline helps LLM accurately understand the instruction text of security experts and achieve effective traffic pattern learning in different downstream tasks, forming generalized representation across different tasks. *(iii) Extensible adaptation with parameter-effective fine-tuning (EA-PEFT)*: To facilitate LLM's generalization to new environments, TrafficLLM employs extensible adaptation using parameter-effective fine-tuning (PEFT) technique [43]. EA-PEFT introduces additional parameters to split task understanding and downstream pattern learning into various PEFT models, which helps TrafficLLM preserve existing capabilities and upgrade model on new network environments.

We build TrafficLLM prototype to achieve model dialogue and pattern reasoning on ten downstream tasks, which supports traffic analysis for different applications (e.g., mobile apps, websites, and malware), protocols (e.g., HTTP, TLS1.3, and DoH), network environments (e.g., VPN, Tor, and botnet), and threats (e.g., web attacks and APT attacks). Through moderate learning, TrafficLLM build robust representation for two key abilities, i.e., traffic detection and traffic generation ability, to assist traffic analytics in their daily attack detection and simulation work, with 5.90%-80.12% and 3.07%-33.92% of performance improvement. We further evaluate TrafficLLM on unseen environments and real-world scenarios. Results show stronger generalization compared to existing ML models.

Contributions. Our contributions can be shown as follows:

- We for the first time explore adapting LLM for traffic representation learning. We systematically analyze the feasibility of realizing the LLM adaptation for achieving strong generalization across diverse traffic analysis tasks.
- We propose TrafficLLM, an adaptation framework to leverage LLM for traffic analysis. TrafficLLM employs traffic-domain tokenization to mitigate the modality gap and generalize to heterogeneous data, dual-stage tuning pipeline to conduct multimodal learning across diverse tasks, and EA-PEFT to realize generalization to new environments.
- We construct the first large-scale traffic-domain LLM adaptation dataset for future research. To the best of our knowledge, we have collected the largest LLM fine-tuning dataset for the traffic domain to date, which includes over 0.4M samples consisting of instruction text and traffic data supervised by experts and AI assistants.
- We conduct extensive experiments on various downstream tasks to demonstrate the superiority of TrafficLLM. TrafficLLM can achieve better performance with robust representation compared to 15 state-of-the-art traffic detection or generation methods. Moreover, TrafficLLM obtains strong generalization on unseen data and real-world settings.

Source codes and datasets. TrafficLLM's source codes and tuning datasets are all available at https://github.com/ZGC-LLM-Safety/TrafficLLM. We discuss the open science policy and ethical considerations in Appendix A.

2 Preliminaries

2.1 Problem Statement

In this paper, we aim to exploit LLM to facilitate the work of network traffic analysis with strong generalization across different tasks. Given the language instructions that involve diverse task descriptions and traffic data that contains multi-type benign or malicious traffic, an adapted LLM in the traffic domain is expected to learn robust traffic representation with its accuracy and generalization in pattern learning. Note that *all work is completed only through model dialogue*, which reduces the operational threshold and accelerates analysis efficiency for security practitioners. However, the characteristics of the traffic domain mainly leave three limitations to realizing LLM's generalization on traffic analysis tasks.

Limitation 1: Generalization to heterogeneous input of traffic data. Traffic data consists of structured metadata in packets (e.g., IPs and ports) and statistical features in flows (e.g., packet lengths and time intervals in flows). However,

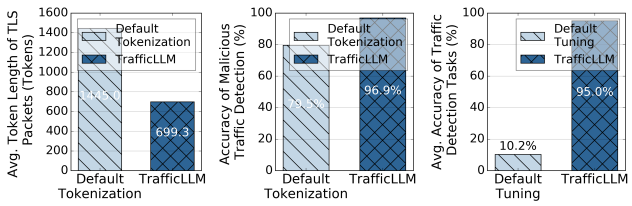


Figure 1: Native LLM's limitation to handle traffic data with default tokenization and tuning strategies. Left and Middle: LLM is ineffective and inaccurate in loading traffic data with language tokens directly. Right: LLM suffers from learning multi-type semantics and traffic features at the same stage.

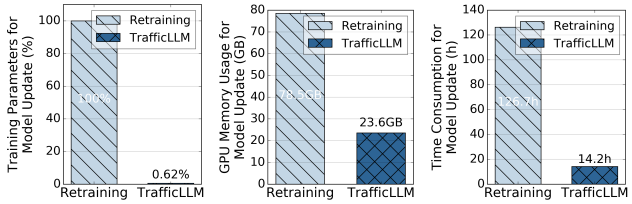


Figure 2: High adaptation costs of LLM's retraining to update traffic detection capabilities on new scenarios. TrafficLLM leverages EA-PEFT to reduce the overhead by using multiple external parameters to encapsulate different capabilities.

most LLMs are considered as the specialized model for processing plain text, which has a huge gap from the traffic data. Before being fed into LLM, the text is converted into language tokens using a standard tokenizer [53, 74]. These tokenizers are usually trained on large-scale text corpora with tokenization algorithms like WordPiece [66] and Byte-Pair Encoding (BPE) [61], which rarely see heterogeneous traffic data. Consequently, LLM may fail to directly transform traffic data into textual formats and load them with default tokenization.

To give an instance, we adapt Llama2-7B [74] to conduct the malware traffic detection (MTD) [80] task with its native tokenizer. First, it is not effective to split traffic information as the input. As shown in Figure 1 (left), the default tokenizer will produce many redundancies when processing metadata in TLS packets. It may reduce the efficiency of LLMs in realistic traffic analysis work since traffic detection tasks require high real-time reliability. Second, the transformed tokens are not performed well to ensure detection accuracy. In Figure 1 (middle), the performance of native LLM is not remarkable on the MTD task (only 79.5% of accuracy on USTC-TFC 2016 dataset [80]) since the unsuitable tokenization split key features incorrectly, leading to the failure to capture distinct patterns between benign and malicious traffic.

Limitation 2: Generalization across different tasks with multimodal learning. Network traffic analysis covers a wide range of specific tasks, such as detecting and simulating attack traffic in different scenarios (e.g., the MTD task). It involves diverse task-specific knowledge in the instruction to prompt LLM to conduct different work. Moreover, these downstream tasks usually point to different network environments, which involves patterns of multi-type traffic features (e.g., packet length sequences in encrypted application classification (EAC) [41] and HTTP request headers in web attack detection (WAD) [38]). These complexities of instructions and traffic patterns can easily confuse LLM when facing multimodal learning [27, 88]. As depicted in Figure 1 (right), we train Llama2 across three traffic detection tasks (i.e., MTD, EAC, and WAD tasks) with the default tuning strategy. Llama2 only reaches 10.2% of average accuracy, indicating the difficulty for LLM to learn with multimodal across different traffic analysis work.

Limitation 3: Generalization to new environments with model update. The adaptation cost of LLM is quite expensive as it requires training large-scale parameters with extensive datasets [9, 53]. However, many traffic analysis tasks often need to update models to struggle with dynamic scenarios, which are raised by the application version updates (e.g., concept drift [35, 85]) and attack method changes (e.g., APT attacks [50]). The high adaptation costs of LLMs prevent the update of model capabilities on new scenarios. As shown in Figure 2, we measure Llama2-7B's adaptation overhead on 5 NIVIDA A800-80GB GPUs for traffic detection tasks. Traditional retraining methods consume 78.5GB GPU memory and 126.7h to adapt to new environments for one epoch, which is unacceptable in real-world dynamic scenarios.

2.2 Investigation on Existing LLMs

We survey existing LLM's capabilities for network traffic analysis. Due to the difficulty of processing traffic data, we have not witnessed the deployment of the foundation model for network traffic analysis in the industry. As of August 19, 2024, we have compiled the model capabilities of the current mainstream LLMs in Table 1 and Appendix B. Although the existing LLMs have initially possessed certain knowledge in the network security field [90], the majority of them can not accomplish traffic analysis tasks due to the lack of capabilities for traffic processing. Most LLMs lack insights from traffic data learning. They can only respond to basic instructions with inaccurate conclusions. To overcome the problem, a series of works like ET-BERT [40] and PERT [29] have been developed in recent years to process traffic data with pre-trained language models (PLMs) [90], aiming to build effective traffic detection and generation capabilities. However, they have several shortcomings:

- **Development cost.** These approaches mainly utilize pre-training techniques [19], which has a high training time and resource cost. Compared with fine-tuning, they can not inherit existing LLMs' abilities, which is less practical.
- **Model size.** These models are usually smaller than 1B. Strictly speaking, they are not part of LLMs [90]. These models may lose LLM's surprising emergent abilities [60,

82] with strong generalization that can not be present in small models.

- **Applications.** These efforts only train on traffic datasets. They fall short in handling natural language, rendering them incapable of following instructions and conducting complex traffic analysis tasks [42].
- **Abilities.** These methods have shortcomings in traffic detection and generation. They do not have the generalization abilities to detect traffic on unseen data [23]. Moreover, they can only generate 5-tuples of packets and flows, whose practical uses are very limited [45, 78].

2.3 Threat Model

TrafficLLM is proposed to develop an LLM for traffic representation, which can be leveraged to construct traffic detection and generation methods to replace traditional ML-based methods that can be integrated into existing sophisticated network traffic analyzers (NTA) [67] and security information and event management (SIEM) systems [5], which are widely deployed in Security Operation Centers (SOCs) to analyze abnormal events based on traffic mirroring and logs [14].

Different from the threat models in the existing ML-based traffic detection and generation studies [22, 23, 75, 86], we consider that TrafficLLM can be directly driven by instructions from experts to accomplish a wide range of downstream tasks. Based on LLM's pattern learning and generalization abilities, we develop TrafficLLM to build the centralized traffic analysis solution, which can achieve the following goals:

- **Threat detection.** TrafficLLM establishes comprehensive traffic detection capabilities to process and analyze diverse benign and malicious traffic. Learning with a variety of traffic data, TrafficLLM can extract traffic features at flow [41, 75] or packet level [44] to identify benign and malicious categories, or realize more fine-grained classification (e.g., application types in encrypted application classification (EAC) [40] and network types in botnet detection (BND) [7]).
- **Attack simulation.** TrafficLLM can generate attack samples to facilitate red teaming and enhance the robustness of network-based IDSes [31, 48]. Different from existing ML-based traffic generation studies [45, 78, 86], TrafficLLM can generate a wide range of target traffic with `.pcap` format based on LLM's strong memorization [11]. It can help security practitioners measure the vulnerability of systems and build robust IDSes through data augmentation.

3 Design of TrafficLLM

3.1 Overall Framework

We present the architecture overview of TrafficLLM in Figure 3, which is built upon a sophisticated fine-tuning framework using natural language and traffic data. Specifically, we mainly propose the following techniques in TrafficLLM to enhance the utility of large language models in network traffic analysis:

Table 1: The basic information and model capabilities of current mainstream LLMs and traffic-domain PLMs. ("✔" = has the ability. "✖" = doesn't have the ability. "○" = has the basic ability but still has shortcomings)

Model	Basic Information		Model Capability		
	Model Size	Open Source	Traffic Detection	Traffic Generation	Language Processing
Llama3 [47]	8B/70B	Yes	✖	✖	✔
Gemini1.5 [26]	Unk.	No	✖	✖	✔
Claude3 [4]	Unk.	No	✖	✖	✔
Mistral Large 2 [34]	Unk.	No	✖	✖	✔
GPT-4 [53]	Unk.	No	✖	✖	✔
GLM-4 [87]	9B/130B	Yes	✖	✖	✔
Baichuan4 [1]	53B	Yes	✖	✖	✔
ET-BERT [40]	0.1B	Yes	✔	✖	✖
PERT [29]	0.04B	Yes	✔	✖	✖
netFound [27]	0.2B	No	✔	✖	✖
NetGPT [45]	0.1B	No	✔	○	✖
Lens [78]	0.25B	No	✔	○	✖
TrafficLLM	6B/12B	Yes	✔	✔	✔

Traffic-Domain Tokenization. To overcome the modality gap between natural language and heterogeneous traffic data, TrafficLLM introduces traffic-domain tokenization to process the diverse input of traffic detection and generation tasks for LLM adaptation. This mechanism effectively extends LLM's native tokenizer by training specialized the tokenization model on large-scale traffic-domain corpora (Section 3.2).

Dual-Stage Tuning Pipeline. TrafficLLM employs a dual-stage tuning pipeline to achieve LLM's robust representation learning across different traffic-domain tasks. The pipeline trains LLM to understand instructions and learn task-related traffic patterns at different stages, which builds upon TrafficLLM task understanding and traffic reasoning abilities for diverse traffic detection and generation tasks (Section 3.3).

Extensible Adaptation with Parameter-Effective Fine-Tuning (EA-PEFT). To adapt LLM for generalization to new traffic environments, TrafficLLM proposes an extensible adaptation with parameter-effective fine-tuning (EA-PEFT) to update model parameters with low overhead. The technique splits model capabilities in different PEFT models, which helps minimize the adaptation costs on dynamic scenarios raised by traffic pattern changes (Section 3.4).

3.2 Traffic-Domain Tokenization

TrafficLLM introduces a detailed strategy to encode original inputs of traffic analysis tasks and makes them learnable for LLMs. This strategy is achieved by two core steps: (i) extracting tuning data and (ii) training the traffic-domain tokenizer.

Tuning Data Extraction. To help LLMs reduce the modality gap and make them feasible to handle traffic data, TrafficLLM first designs specialized extractors to extract suitable traffic

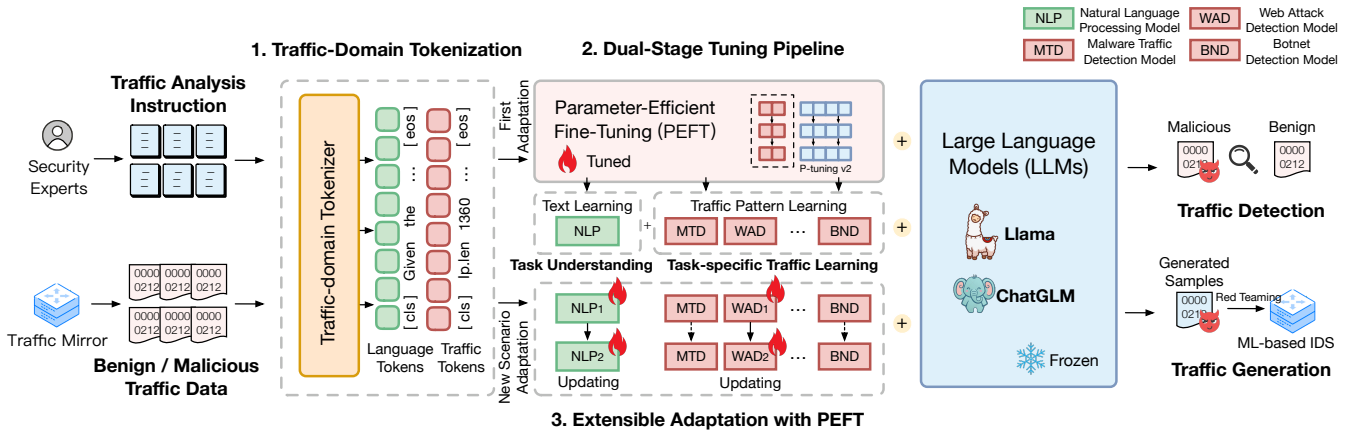


Figure 3: The overall framework of TrafficLLM. TrafficLLM employs three core techniques: *traffic-domain tokenization* to process instructions and traffic data, *dual-stage tuning pipeline* to understand text semantics and learn traffic patterns across different tasks, *extensible adaptation with parameter-effective fine-tune* to update model parameters for new scenario adaptation.

features for different tasks. Based on the experience of previous work on traffic detection and generation [38, 65, 75, 86], we consider their feature extraction outcomes are still orthogonal to TrafficLLM. For instance, BIND [2] and FlowPrint [75] extracts statistical features for encrypted traffic classification. RETSINA [38] extracts HTTP request headers to conduct effective web attack detection. TrafficLLM can reuse these advanced extraction methods to extract HTTP request headers, packet direction sequences, and statistical features for WAD tasks, WF tasks, and other traffic detection tasks, which keeps generalization to diverse traffic data (see Appendix C).

Traffic Detection Tuning Data Example

Instruction: Given the following traffic data that contains protocol fields, traffic features, and payloads. Please conduct the Malicious Traffic Detection task to determine which application category the encrypted benign or malicious traffic belongs to.
<packet>: ip.len: 1360, ip.proto: 6, tcp.srcport: 443, tcp.dstport: 56603, tcp.len: 1308, tcp.window_size: ...
Output: Zeus .

Traffic Generation Tuning Data Example

Instruction: Based on the protocol fields, traffic features, and payloads of traffic in your knowledge. Please generate a packet of Skype traffic .

Output: Skype.pcap (ip.src: 1.2.102.211, ip.dst: 1.1.210.113, tcp.srcport:443, tcp.dstport: 27567, raw: 1021ac5010000021ac50200000800450002aeea10400 0200632f2010266d30101...).

To facilitate LLMs to achieve remarkable traffic pattern learning abilities, TrafficLLM propose prompt-learning [42] to build tuning data templates and adapt LLMs to the traffic-domain semantic space. We utilize Tshark [21] to extract features in different packet layers. The features are organized by pairs including the field name and the corresponding value (e.g., `tcp.srcport: 443`). To satisfy different granularity for traffic pattern learning (i.e., packet and flow levels) and indicate the beginning of traffic data, we define an indicator token `<packet>` in the context. Each packet data is started with this special indicator to form the flow data. These prompt-learning methods help LLM capture valuable semantics for pattern learning. Finally, TrafficLLM combines traffic analysis instructions and features to build the tuning data. The examples of TrafficLLM's tuning data are shown above.

Tokenizer Training. After extracting the tuning data, we build a specialized traffic-domain tokenizer to form the input traffic tokens. We use the BPE [61] method to train the specialized tokenizer on the large-scale tuning data. Since native LLM has hardly ever seen traffic data, it can be considered as an extension to the existing tokenizer. Table 2 shows an example of tokenization of TrafficLLM and an open-sourced LLM ChatGLM2 [87]. Tokenizers of existing LLMs tend to split the field names (e.g., `checksum` and `tcp`) since they have learned less of the traffic-domain languages. In contrast, TrafficLLM's tokenizer can retain these feature indicators based on their appearance frequency in training data. Furthermore, the traffic-domain tokenizer can also store the common ports, network segments, and flags, helping LLMs learn the numerical features correctly. Due to the accurate tokenization on traffic word features, TrafficLLM can produce shorter packet tokens with a 699.36 average token length, compared to ChatGLM2's 1445.04 token length.

As shown in Figure 1 (left) and (middle), TrafficLLM's tokenization is more effective and accurate in loading traffic data compared to the default tokenization. This mechanism helps TrafficLLM reach 106% efficiency improvement to process traf-

Table 2: The tokenization of TrafficLLM's tokenizer on the traffic word features compared to ChatGLM2's tokenizer.

Default tokenization: _ip . proto : _ 6 , _ip . che cks um : _ 0 x 0 0 0 0 1 7 a 7 , _ip . che cks um . status : _ 2 , _ip . src : _ 1 0 . 0 . 2 . 1 5 , _ip . d st : _ 1 9 8 . 5 2 . 2 0 0 . 3 9 , _t cp . src port : _ 4 3 7 3 1 , _t cp . d st port : _ 4 4 3 , _t ... [Token length : 1405]
TrafficLLM tokenization: _ip . proto : _6, _ip . checksum : _0 x 000017 a 7, _ip . checksum . status : _2, _ip . src : _10.0. 2.15, _ip . dst : _198.52. 200.39, _tcp . srcport : _43731, dstport : _443, ... [Token length : 690]

fic data. TrafficLLM also achieves 17.4% better performance on MTD tasks by using the traffic-domain tokenization.

3.3 Dual-Stage Tuning Pipeline

After building the training data and the specialized traffic-domain tokenizer, we focus on two challenges of tuning LLMs: (i) How to help LLMs understand the task-related natural language to determine which task should be conducted; (ii) How to learn the task-specific traffic pattern across different tasks. To address these, TrafficLLM proposes a dual-stage tuning pipeline to help LLMs achieve effective multimodal learning on diverse traffic detection and generation tasks.

Tuning Objectives. The core consideration of applying LLM is to learn traffic representations with strong generalization. Based on LLM's strong memorization [11] coming from the deep Transformer [76] architecture, these representations can obtain distinct traffic patterns, e.g., the low-rate and diverse traffic patterns of encrypted malicious traffic [32, 72]. TrafficLLM mainly uses the robust representation to realize two mainstream tasks, i.e., *traffic detection* and *traffic generation*.

- **Traffic detection.** Given the security expert's instruction $S = \{s_1, s_2, ..., s_m\}$ that contains m language tokens, the traffic data $X = \{x_0, x_1, ..., x_n\}$ that contains n traffic tokens to describe the traffic features, *traffic detection* requires the task-related instruction S_i and traffic data X_i (flows or packets) as TrafficLLM's input (S_i, X_i). Then, TrafficLLM can identify the ground truth label $y_i \in Y = \{y_0, y_1, ..., y_c\}$ of the traffic across different traffic detection tasks (e.g., MTD, WAD, and BND tasks) with its parameter θ:

$$y_i = \text{TrafficLLM}((S_i, X_i)|\theta) \tag{1}$$

- **Traffic generation.** *Traffic generation* can be regarded as the reversed process of traffic detection tasks. It aims to input the generation instruction S_i to describe the specific scenario and the traffic category $y_i \in Y = \{y_0, y_1, ..., y_c\}$ of the traffic to be simulated. TrafficLLM can generate a synthetic packet $\hat{X}_i$ that satisfies the instruction:

$$\hat{X}_i = \text{TrafficLLM}((S_i, y_i)|\theta) \tag{2}$$

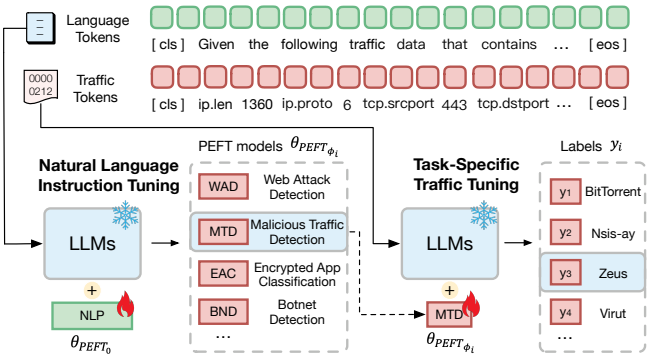


Figure 4: Illustration of the dual-stage tuning pipeline to learn natural language and traffic patterns respectively.

Natural Language Instruction Tuning. In the first stage of the dual-stage tuning in TrafficLLM, we introduce natural language instruction tuning to inject the professional task description text from the field of cybersecurity into LLMs. As shown in Figure 4, in this mechanism, we force LLM to understand the instructions from security experts and predict the task name ϕ_i that needs to be performed.

$$\phi_i = \text{TrafficLLM}_{stage_1}(S_i|\theta_1) \tag{3}$$

where θ_1 is the trainable parameters in the first stage. ϕ_i is the downstream task name to be performed. To learn the context of the security task description, we follow LLM's autoregressive objective function to converge the model. Given the human instruction $S_i = \{s_1, s_2, ..., s_m\}$, TrafficLLM calculates the probability P_m of token s_i to model the loss $\mathcal{J}(\theta_1)$:

$$\begin{aligned} P_m(s_i|s_1, ..., s_{i-1}) &= \text{softmax}(W_s h_{i-1}) \\ \mathcal{J}(\theta_1) &= \sum_i \log P_m(s_i|s_1, ..., s_{i-1}) \end{aligned} \tag{4}$$

where W_s is the learning parameter matrix for task understanding. h_{i-1} denotes the representation encoded in TraficLLM with the input of the preceding $i-1$ tokens. The natural language instruction tuning technique plays a crucial role in accurately matching instruction text with the corresponding downstream task, thereby enabling LLMs' domain knowledge to understand different traffic analysis tasks.

Task-Specific Traffic Tuning. The second stage we propose is task-specific traffic tuning. After understanding the task, we force TrafficLLM to learn the traffic pattern under the downstream tasks. In this stage, we fine-tune LLM using the training pairs (X_i, y_i) to model the traffic representation under traffic detection tasks ϕ_{TD} and traffic generation tasks ϕ_{TG}. For the specific downstream task ϕ_i, TrafficLLM trains the second stage parameters θ_2 to predict traffic labels y_i or generate

synthetic traffic $\hat{X}_i$:

$$y_i = \text{TrafficLLM}_{stage2}(X_i|(\theta_2;\phi_i \in \phi_{TD}))$$
$$\hat{X}_i = \text{TrafficLLM}_{stage2}(y_i|(\theta_2;\phi_i \in \phi_{TG})) \quad (5)$$

To inject the knowledge of traffic features into LLMs, TrafficLLM build the representation of traffic data $X_i = \{x_1, x_2, ..., x_n\}$, by learning the context of traffic flow or packet features with the loss function $\mathcal{J}(\theta_2)$:

$$P_n(x_i|x_1, ..., x_{i-1}) = \text{softmax}(W_l h_{i-1})$$
$$\mathcal{J}(\theta_2) = \sum_i \sum \log P_n(x_i|x_1, ..., x_{i-1}) \quad (6)$$

where W_l is the trainable parameter matrix for traffic pattern learning. The task-specific traffic tuning aims to align the LLMs with various traffic data under different scenarios, such as VPN and Tor networks, which allows LLMs to accomplish diverse downstream tasks in our framework.

As shown in Figure 1 (right), the dual-stage tuning pipeline helps TrafficLLM achieve 95.0% of average accuracy across MTD, EAC, and WAD tasks. TrafficLLM has an 84.8% higher accuracy than direct fine-tuning by learning text semantics and task-specific traffic patterns in stages.

3.4 Extensible Adaptation with Parameter-Effective Fine-Tuning (EA-PEFT)

To realize LLM's generalization to new traffic environments, TrafficLLM employs extensible adaptation with parameter-effective fine-tuning (EA-PEFT) to efficiently update LLMs on dynamic scenarios. The EA-PEFT mainly includes: (i) Parameter effective fine-tuning to achieve traffic domain adaptation with less overhead; (ii) Extensible adaptation to align LLM with new environments.

Traffic Domain Adaptation with PEFT. Assume that the pre-trained LLM's parameters are θ_{LLM}, to adapt to new environments, traditional retraining methods require training the full parameters of models, which indicates the parameter update $\Delta\theta$ is equal to θ_{LLM}, i.e., $|\Delta\theta| = |\theta_{LLM}|$. However, the large parameter size of LLM entails huge costs of repeated retraining in new environments. To address this limitation, TrafficLLM freezes the parameters of the LLM and tunes extra parameters to achieve parameter effective fine-tuning (PEFT) [20]. During the dual-stage tuning, TrafficLLM tunes additional parameters to respectively build PEFT models θ_{PEFT_0} for task understanding and $\theta_{PEFT_{\phi_i}}$ for task-specific traffic learning:

$$\theta_1 = \theta_{LLM} + \theta_{PEFT_0}$$
$$\theta_2 = \theta_{LLM} + \theta_{PEFT_{\phi_i}} \quad (7)$$

where θ_{PEFT_0} and $\theta_{PEFT_{\phi_i}}$ are the parameter updates $\Delta\theta$ in the two stages. This strategy helps TrafficLLM encapsulate abilities of natural language processing and traffic pattern

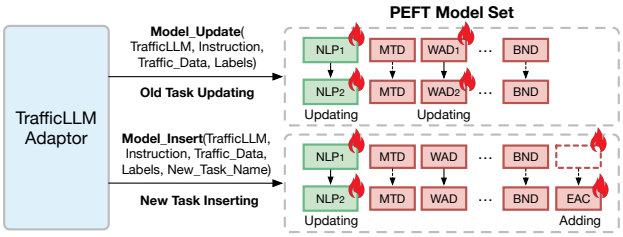


Figure 5: The workflow of the extensible adaptation with parameter-effective fine-tuning (EA-PEFT) in TrafficLLM.

learning across different tasks into specialized PEFT models, which are triggered by instructions from different tasks.

Extensible Adaptation with PEFT Models. Using the PEFT models trained during traffic domain adaptation, TrafficLLM employs EA-PEFT to organize these models with an extensible adaptation, which can help TrafficLLM easily adapt to new environments. Figure 5 shows the overview of EA-PEFT's workflow implemented by Python scripts. In EA-PEFT, TrafficLLM adaptor allows flexible operations to update old models or register new tasks. For instance, When faced with the traffic update in WAD tasks raised by client version upgrade (e.g., App version drift [35]) or attack method changes (e.g., HTTP request body changes [38]), the adaptor can call `Model_Update` to update the specific PEFT models by providing new EAC or WAD datasets. Moreover, TrafficLLM can easily add new traffic analysis scenarios. The adaptor can schedule `Model_Insert` to train new PEFT models and insert them in the EA-PEFT framework. Based on these, TrafficLLM easily scales to a wide range of traffic domain tasks with the light-wise adaption scheme of EA-PEFT.

As shown in Figure 2, the EA-PEFT technique helps TrafficLLM only need to train 0.62% of parameters when adapting to new traffic environments, which significantly reduces the adaptation costs with the reduction of 69.9% GPU memory and 88.8% training time. TrafficLLM effectively mitigates the high adaptation cost of retraining methods, facilitating deploy more traffic analysis tasks in real-world settings.

4 Experiment Setup & Implementation

In this section, we introduce the experimental setup and implementation of TrafficLLM, including the dataset, baselines, metrics, and implementation settings used in the experiments.

4.1 Dataset

To comprehensively evaluate the effectiveness of the TrafficLLM, we collect a wide set of traffic datasets and natural language instructions for LLM adaptation.

Traffic Datasets. The traffic datasets used in our experiments are shown in Table 3. We use 10 traffic datasets with different

Table 3: The detail of 10 traffic datasets used to build the traffic analysis downstream tasks in experiments. TrafficLLM use the traffic data and labels of these datasets to build traffic detection and generation capabilities respectively.

Tasks	Abbrev.	Traffic Datasets	Description	#Flows	#Packets	#Labels
Malware Traffic Detection	MTD	USTC TFC 2016 [80]	10-class malware and 10-class benign Apps	9,853	97,115	20
Botnet Detection	BND	ISCX Botnet 2014 [7]	4-class botnets and 1-class benign network	30,511	300,000	5
Malicious DoH Detection	MDD	CIC DoHBrw 2020 [49]	4-class benign DoH and 1-class malicious DoH	545,463	28,341,000	5
Web Attack Detection	WAD	CSIC 2010 [16]	1-class Web attack requests and 1-class benign requests	-	61,000	2
APT Attack Detection	AAD	DAPT 2020 [50]	1-class APT attack traffic and 1-class benign traffic	3,000	10,000	2
Encrypted VPN Detection	EVD	ISCX VPN 2016 [25]	19-class VPN encrypted App traffic	3,694	60,000	14
Tor Behavior Detection	TBD	ISCX Tor 2016 [37]	8-class user behaviors under Tor network	3,021	80,000	8
Encrypted App Classification	EAC	CSTNET 2023 [40]	20-class mobile App traffic using TLS encryption	65,128	602,568	20
Website Fingerprinting	WF	CW-100 2018 [59]	100-class website accessing traffic under Tor	9,000	-	100
Concept Drift	CD	APP-53 2023 [35]	53-class mobile App traffic with concept drift	133,000	449,000	53

Table 4: The statistic information and the top nouns of the natural language instructions we collect for task understanding.

Statistics	Value	Statistics	Value	Word	%Hits	Word	%Hits	Word	%Hits
Total words	128,248	Average number of words per instruction	15.26	traffic	4.15%	packet	1.01%	software	0.48%
Total unique words	1,999	Average number of unique words per instruction	13.92	network	2.58%	application	0.79%	tunnel	0.44%
Total sentences	15,238	Average number of sentence per instruction	1.65	data	1.60%	IP	0.56%	behavior	0.35%
Total instructions	9,209	Type Token Ratio (TTR)	1.56	field	1.54%	botnet	0.49%	set	0.33%

task scenarios to collect $\approx$ 0.4M training data, which helps build LLMs' traffic detection and generation capabilities:

- **Traffic Detection.** To evaluate the detection performance of TrafficLLM on various network scenarios, we choose 10 traffic datasets to measure TrafficLLM's abilities to detect *malicious and benign traffic*. In *malicious traffic detection tasks*, we introduce malware traffic detection (USTC TFC 2016 [80]), botnet detection (ISCX Botnet 2014 [7]), malicious DoH detection (CIC DoHBrw 2020 [49]), web attack detection (CSIC 2010 [16]), and APT attack detection (DAPT 2020 [50]) tasks. In *fine-grained benign traffic detection*, we employ encrypted VPN detection (ISCX VPN 2016 [25]), Tor behavior detection (ISCX Tor 2016 [37]), encrypted App classification (CSTNET 2023 [40]), website fingerprinting (CW-100 2018 [59]), and concept drift (APP-53 2023 [35]) scenarios. We use task-related traffic detection instructions, flows/packets in the traffic datasets, and the corresponding labels to build the human instructions S_i, traffic data X_i, and the target label y_i.
- **Traffic Generation.** To implement the packet simulation capability, we reuse the 10 task-specific traffic datasets mentioned above to build the traffic generation datasets. To design the training data, we use the traffic label y_i to produce the generation task instruction S_i and sample the packets to form the target synthetic packet $\hat{X}_i$.

Note that we choose these datasets to cover various applications (i.e., mobile apps, websites, and malware), protocols (i.e., HTTP, QUIC, TLS1.3, and DoH), network environments (i.e., VPN, Tor, and botnet), and attacks (i.e., web attacks and APT attacks) in the traffic, which have good variety to evaluate TrafficLLM's robustness across different scenarios.

Natural Language Instructions. We show the details of the natural language dataset in Table 4. To build the natural language corpus as the human instructions in TrafficLLM, we invite security experts and college students to provide accurate task descriptions for each downstream task. Moreover, to increase the diversity of the context, we use ChatGPT [54] to rewrite these expert prompts through prompt engineering and remove similar instructions based on human annotation. Each instruction is rewritten 20 times at least. Finally, we collect 9,209 text instructions to build the training data (More details of the instruction settings can be seen in Appendix D.1).

4.2 Baselines & Evaluation Metrics

Baselines. To measure the performance of TrafficLLM, we mainly use two types of baselines, including ML-based traffic detection methods and traffic generation methods (More detailed settings are shown in Appendix D.2):

- **ML-based detection methods.** We use state-of-the-art traffic detection methods across different tasks as baselines to evaluate the traffic detection in TrafficLLM. The baselines include (i) Statistical Feature Methods: AppScanner [69], CUMUL [55], BIND [2], k-fingerprinting (K-FP) [28], and FlowPrint [75]; (ii) Deep Learning Methods: FS-Net [41], Deep Fingerprinting (DF) [65], GraphDApp [62], TSCRNN [39], and Deeppacket [44]; (iii) Pre-training Methods: PERT [29] and ET-BERT [40].
- **ML-based generation methods.** To evaluate the performance of traffic generation, we compare TrafficLLM to state-of-the-art ML-based generation methods. The baselines include (i) The GAN-based IP header trace generation algorithm: Netshare [86]; (ii) The conditional GANs-based augmentation method: PacketCGAN [12]; (iii) The CNN-GAN-based IP packet generator: PAC-GAN [77]. Note that

Table 5: Traffic detection results on CW-100 2018, APP-53 2023, ISCX Tor 2016, and ISCX VPN 2016, and CSTNET 2023.

Dataset	ISCX Tor 2016			ISCX VPN 2016			APP-53 2023			CSTNET 2023			CW-100 2018		
Method	PR	RC	F1	PR	RC	F1	PR	RC	F1	PR	RC	F1	PR	RC	F1
AppScanner [69]	0.7251	0.6512	0.6124	0.7395	0.7125	0.7304	0.7035	0.6957	0.6980	0.6481	0.6420	0.6467	-[1]	-	-
CUMUL [55]	0.5672	0.5731	0.5628	0.6322	0.6824	0.6570	0.5563	0.5467	0.5480	0.5373	0.5217	0.5274	-	-	-
BIND [2]	0.4569	0.4385	0.4469	0.5067	0.4975	0.5008	0.6566	0.6456	0.6502	0.7712	0.7689	0.7691	-	-	-
K-FP [28]	0.7035	0.6789	0.6951	0.6784	0.6967	0.6891	0.5660	0.5260	0.5295	0.4172	0.3981	0.4012	-	-	-
FlowPrint [75]	0.4201	0.3789	0.3901	0.7084	0.6608	0.6888	0.4890	0.5023	0.4950	0.2371	0.2270	0.2254	-	-	-
GraphDApp [62]	0.4789	0.4878	0.4781	0.6478	0.6488	0.6476	0.6860	0.6450	0.6550	0.6329	0.5965	0.6078	-	-	-
FS-Net [41]	0.6283	0.6274	0.5916	0.7693	0.7488	0.7507	0.8550	0.8349	0.8376	0.8291	0.8061	0.8195	0.4582	0.4781	0.4668
DF [65]	0.6072	0.6123	0.6090	0.6296	0.6051	0.6139	0.7689	0.7523	0.7604	0.7729	0.7621	0.7682	0.9120	0.9046	0.9075
TSCRNN [39]	0.9051	0.9178	0.9105	0.9346	0.9367	0.9349	0.7057	0.6890	0.6995	0.7529	0.7566	0.7558	0.8350	0.8210	0.8260
Deeppacket [44]	0.7456	0.7469	0.7400	0.9467	0.9508	0.9503	0.5590	0.5489	0.5506	0.4013	0.2965	0.3890	0.8243	0.8246	0.8244
PERT [29]	0.7480	0.4952	0.4874	0.8573	0.7394	0.7481	0.8458	0.8369	0.8403	0.8896	0.8721	0.8771	0.8247	0.8355	0.8300
ET-BERT [40]	0.9186	0.9430	0.9368	0.9567	0.9420	0.9539	0.8540	0.8494	0.8506	0.9581	0.9478	0.9496	0.8670	0.8650	0.8660
TrafficLLM (packet)	0.9810[2]	0.9871	0.9810	0.9960	0.9970	0.9960	0.9325	0.9315	0.9320	0.9678	0.9369	0.9599	-	-	-
TrafficLLM (flow)	0.8756	0.8778	0.8759	0.9880	0.9878	0.9879	0.8850	0.8765	0.8805	0.9485	0.8490	0.8855	0.9370	0.9360	0.9366

[1] - means that the dataset formats are not suitable for the methods. Statistical feature methods and GraphDApp can not handle CW-100 2018 and CSIC 2010 datasets.
[2] We highlight the best performance in • and the second best performance in • for each metric under different datasets.

the rule-based methods are not in the range since they can only simulate network characteristics and are difficult to generate fine-grained features of packets with manual configuration (e.g., diverse patterns in encrypted App traffic).

Evaluation Metrics. We use the following metrics to evaluate the traffic detection and generation abilities: (i) Precision (PR), (ii) Recall (RC), (iii) Macro F1-score (F1), (iv) Accuracy (Acc), and (v) the macro-average area under ROC curve (Macro-AUC). We employ similarity metrics (vi) Jensen-Shannon Divergence (JSD) to evaluate the packet simulation performance. Note that lower JSD denotes better fidelity.

4.3 Implementation of TrafficLLM

Testbed & Implementation. We conduct our experiments on a Super GPU server (super-SYS-420GP-TNR) with 5 NVIDIA A100 80G GPUs, Ubuntu 18.04.1 (Linux 5.4.0), and 1TB memory. We use PyTorch 2.0.1 to build the prototype TrafficLLM and deploy Python scripts to incorporate TrafficLLM adaptor and PEFT models in the EA-PEFT framework. We employ Llama2-7B [74] and ChatGLM2-6B [87] as the base LLMs for most experiment. We choose P-Tuning v2 [43] as the PEFT method. The storage of each PEFT model is 7.1MB.

Hyper-parameters. During the data pre-processing, we employ a data sampling process for each class to avoid the data imbalance issue. The maximum number of packets or flows in each class is 5,000. We set the ratio of training sets, validation sets, and test sets to 8:1:1. In traffic detection tasks, we set the maximum packet-feature length and flow-feature length to 1,024 and 2,560 at the packet-level and flow-level detection (The maximum length of each packet is 256 in the flow-level detection). During the training stage, we set the training steps as 20,000; the initial learning rate is 2×10^{-2}. The maximum source and target lengths of generation tasks are set as 128 and 3,072, while detection tasks are 3,072 and 32.

5 Evaluation

In this section, we implement TrafficLLM based on the above setups and evaluate its performance across different tasks (More evaluation setups are shown in Appendix D.3 and E).

5.1 Traffic Detection

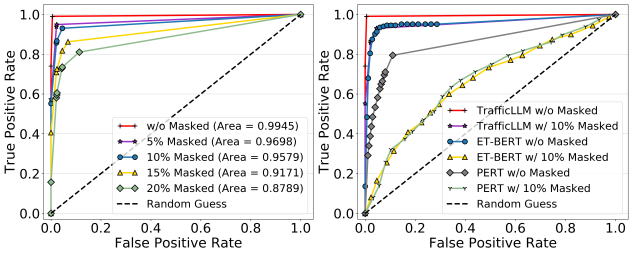
Accuracy on Downstream Tasks. Table 5 and Table 6 present the performance of TrafficLLM on 10 datasets for various downstream detection tasks. Results indicate that TrafficLLM can classify all 229 types of traffic with F1-score ranging from 0.9320 to 0.9992. TrafficLLM achieves at most 80.12% better results than all baselines. Pre-training methods like PERT and ET-BERT obtained acceptable results with the average F1-scores of 0.8128 and 0.9324 since they additionally put the traffic bytes of pre-trained datasets into the model compared to prior works. Nevertheless, since TrafficLLM leverages LLM's pattern mining and generalization abilities, the performance outperforms PERT and ET-BERT with an improvement of 9.63% at most on the F1-score metric.

Furthermore, due to the difference between various detection scenarios, most works keep a poor generalization ability to share their models across different tasks (e.g., FlowPrint keeps F1-scores ranging from 0.2254 to 0.7492 with a variance of 3.396%). Results indicate that previous methods are usually adept at specific tasks but fail in unfamiliar network scenarios. For instance, the state-of-the-art VPN detection model Deeppacket gets an F1-score of 0.9503 on the EVD task (ISCX VPN 2016) but only obtains an F1-score of 0.3890 on the EAC task (CSTNET 2023). Conversely, TrafficLLM outperforms existing methods with an average F1-score of 0.9875 and a variance of 0.018%, compared to the pre-trained model ET-BERT's 0.9324 average F1-score and 0.151% variance.

Compared to Pre-trained Models. Different from previous

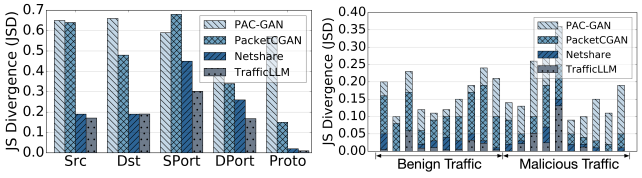
Table 6: Traffic detection results on ISCX Botnet 2014, USTC TFC 2016, DoHBrw 2020, CSIC 2010, and DAPT 2020.

Dataset	ISCX Botnet 2014			USTC TFC 2016			CIC DoHBrw 2020			DAPT 2020			CSIC 2010		
Method	PR	RC	F1	PR	RC	F1	PR	RC	F1	PR	RC	F1	PR	RC	F1
AppScanner [69]	0.9021	0.9004	0.9008	0.8872	0.8910	0.8972	0.7331	0.7063	0.7106	0.7590	0.7226	0.7408	-	-	-
CUMUL [55]	0.8791	0.8320	0.8417	0.6074	0.5239	0.5437	0.5623	0.5281	0.5301	0.6509	0.6486	0.6492	-	-	-
BIND [2]	0.6798	0.6489	0.6582	0.8268	0.8014	0.8209	0.7137	0.7003	0.7077	0.7224	0.7026	0.7115	-	-	-
K-FP [28]	0.8398	0.8960	0.8591	0.6447	0.4172	0.3981	0.7035	0.6789	0.6951	0.6585	0.6496	0.6542	-	-	-
FlowPrint [75]	0.5898	0.6309	0.5967	0.6609	0.6596	0.6584	0.7712	0.2371	0.2270	0.6960	0.6894	0.6935	-	-	-
GraphDApp [62]	0.7578	0.7598	0.7538	0.8027	0.8320	0.8263	0.6478	0.6791	0.6512	0.8350	0.8342	0.8345	-	-	-
FS-Net [41]	0.7303	0.8546	0.7876	0.5964	0.7174	0.6371	0.7123	0.6991	0.7053	0.8056	0.7783	0.7946	0.6255	0.6145	0.6190
DF [65]	0.8267	0.8509	0.7980	0.7623	0.7598	0.7604	0.7078	0.6986	0.7022	0.7892	0.7759	0.7805	0.6470	0.6455	0.6464
TSCRNN [39]	0.9206	0.8976	0.8989	0.9538	0.9428	0.9503	0.8837	0.8672	0.8695	0.8453	0.8550	0.8503	0.6480	0.6326	0.6370
Deeppacket [44]	0.9408	0.9520	0.9496	0.9369	0.9292	0.9338	0.8930	0.8977	0.8965	0.8934	0.8924	0.8930	0.6469	0.6510	0.6505
PERT [29]	0.9268	0.9078	0.9096	0.9605	0.9611	0.9574	0.9378	0.9052	0.8977	0.8990	0.8868	0.8960	0.8274	0.7588	0.7685
ET-BERT [40]	0.9503	0.9462	0.9489	0.9621	0.9508	0.9587	0.8927	0.8674	0.8467	0.9450	0.9423	0.9435	0.9021	0.8920	0.8995
TrafficLLM (packet)	0.9800	0.9861	0.9800	0.9950	0.9957	0.9950	0.9940	0.9940	0.9939	0.9820	0.9806	0.9810	0.9870	0.9823	0.9845
TrafficLLM (flow)	0.9992	0.9992	0.9992	0.9700	0.9710	0.9700	0.8756	0.8778	0.8759	0.9605	0.9595	0.9600	-	-	-



(a) TrafficLLM on Masked Features (b) Compared to ET-BERT and PERT

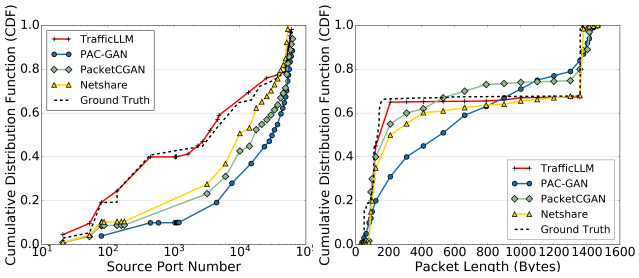
Figure 6: The Macro-AUC of TrafficLLM and baselines with different ratios of masked features on ISCX Tor 2016.



(a) 5-Tuples JSD (b) Destination IP JSD

Figure 7: JSD between real and synthetic distributions on ISCX Botnet 2014 and USTC TFC 2016 (⇓ JSD is better).

pre-trained works [29, 40], a salient property of TrafficLLM is to fine-tune existing LLMs to achieve adaptation in the traffic domain. To prove the superiority, we mask partial features in the inference stage to test detection performance. In Figure 6, results indicate that TrafficLLM can obtain a Macro-AUC of 0.9171 even when 15% of features missing. However, the performance of pre-trained models ET-BERT and PERT significantly declines. For instance, for a target FPR = 1×10^{-1}, while TrafficLLM achieves a TPR of 0.90, both two baselines provide TPRs less than 0.40. The robustness against missing features comes from the pattern reasoning and generalization abilities inherited from LLMs, while previous pre-trained works are only trained on fixed-format datasets, making them fail to reason in changing scenarios like feature missing.



(a) Source port (b) Packet length (bytes)

Figure 8: Source port and packet length CDFs computed with baselines and the ground truth on ISCX VPN 2016.

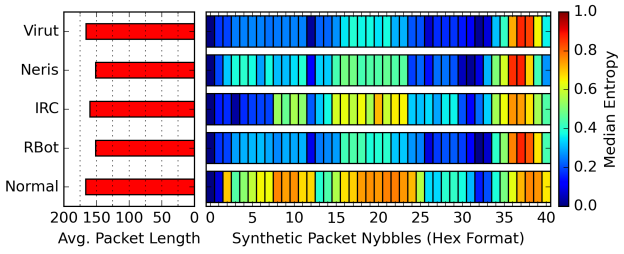


Figure 9: Entropy analysis of synthetic packets on ISCX Botnet 2014. In each index, a higher / lower entropy score (red / blue) refers to more diverse / fixed nybbles generated.

5.2 Traffic Generation

Distribution Metrics Compared to Baselines. To evaluate the performance of the traffic generation capability, we compute the distribution metrics between the real and synthetic distribution of the 5-tuples in the packets. Figure 7 shows the results compared to 3 baselines. We find that TrafficLLM is at most 73.76% better than existing methods to degrade the gap from the real distribution. For different categories of benign and malicious traffic, TrafficLLM achieves an average JSD of 0.0179, which is 39.32% better than the state-of-the-

Table 7: The performance of identifying the 2k synthetic and real samples using the classifiers built from the same size of real and synthetic data under USTC TFC 2016, ISCX Botnet 2014, ISCX VPN 2016, and CSTNET 2023 datasets.

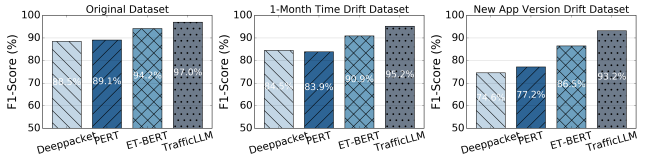
Method	Setting[1]	USTC TFC 2016			ISCX Botnet 2014			ISCX VPN 2016			CSTNET 2023		
		PR	RC	F1	PR	RC	F1	PR	RC	F1	PR	RC	F1
PAC-GAN [12]	❶ R-Train S-Test	0.7825	0.7432	0.7453	0.7234	0.7421	0.7256	0.8196	0.7945	0.8204	0.8402	0.8794	0.8571
PacketCGAN [77]		0.7673	0.7925	0.7529	0.8057	0.8245	0.8050	0.8342	0.8454	0.8345	0.8590	0.8881	0.8530
NetShare [86]		0.8178	0.8157	0.8042	0.9021	0.9105	0.9042	0.9859	0.9475	0.9634	0.9314	0.9146	0.9345
TrafficLLM		0.9315	0.8604	0.8354	0.9778	0.9730	0.9727	0.9902	0.9854	0.9879	0.9861	0.9852	0.9852
PAC-GAN [12]	❷ S-Train R-Test	0.6459	0.6375	0.6431	0.6204	0.6079	0.6202	0.8192	0.8023	0.8205	0.7056	0.6859	0.6980
PacketCGAN [77]		0.6724	0.6458	0.6532	0.7057	0.6894	0.6995	0.8525	0.8678	0.8548	0.6861	0.6404	0.6489
NetShare [86]		0.8521	0.8254	0.8322	0.7974	0.7854	0.8024	0.9045	0.8845	0.8964	0.8420	0.8299	0.8405
TrafficLLM		0.8843	0.8641	0.8589	0.8634	0.8375	0.8334	0.9716	0.9686	0.9688	0.8630	0.8289	0.8340

[1] Setting ❶ uses the real samples to train classifiers to test the synthetic samples. Setting ❷ uses the synthetic samples to train classifiers and test the real samples.

art traffic generation method NetShare's 0.0295 average JSD. For a more detailed analysis, Figure 8 shows the CDFs of the source port and packet length. Results indicate that existing GAN-based methods can not capture the range of fields well. TrafficLLM's memorization advantage from parameter volume can help restore the field values of the original dataset, making the distribution consistent with the ground truth.

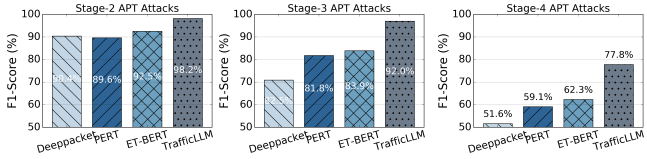
Entropy Analysis on Synthetic Packets. Figure 9 shows the entropy analysis results of the packets simulated from ISCX Botnet 2014. We find that TrafficLLM can learn fixed field values (e.g., TCP Flags) effectively and imitate changed field values (e.g., Source and Destination Port) according to the distribution in the dataset. Moreover, TrafficLLM can capture the traffic characteristics for different network scenarios. For instance, TrafficLLM keeps an average entropy of 0.55 to generate the packet nybbles of the target botnet (i.e., Virut, Neris, IRC, and RBot), while the normal network is 0.64 due to the diversity of protocols in the networks.

Practicality of Synthetic Samples. We consider that the generation capability of TrafficLLM mainly has two promising applications: (i) generating samples for security tests and (ii) building classifiers under few-shot scenarios. First, to verify the quality of the generated samples, we use 2k traffic traces of real datasets to build a robust ensemble model based on Multinomial Naive Bayesian [36] and SGD classifiers [91]. Then we use the synthetic traces to test whether the model can identify these samples. In Table 7, we find that the generated samples of TrafficLLM can be identified by the real-world classifier with an average F1-score of 0.9483, which is 4.68% better than the state-of-the-art method NetShare. It means that TrafficLLM can generate test samples with extremely high confidence. To measure the second application of TrafficLLM, we use 2k generated packets to build the classifiers and treat the real packets as the test sets. Results indicate that TrafficLLM outperforms baselines on most metrics to detect real-world traffic using the classifiers built from synthetic data. TrafficLLM can be applied in data augmentation for benign or malicious traffic with a 0.8739 average F1-score, which is 3.07%-33.92% better than existing ML-based methods.



(a) No Drift (b) Time Drift (c) Version Drift

Figure 10: Concept drift experiments with 1-month time drift and new App version drift settings on APP-53 2023 datasets.



(a) Stage-2 APT Attacks (b) Stage-3 APT Attacks (c) Stage-4 APT Attacks

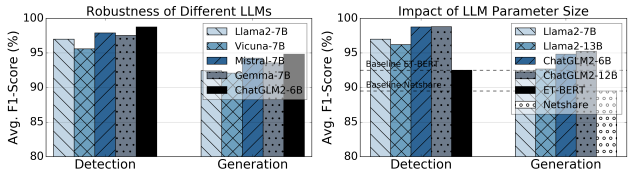
Figure 11: Future stage APT attack detection based on historical stage-1 APT attack knowledge on DAPT 2020 datasets.

5.3 Generalization to Unseen Data

We evaluate TrafficLLM's generalization abilities on unseen data with concept drift and APT attack datasets [35, 50].

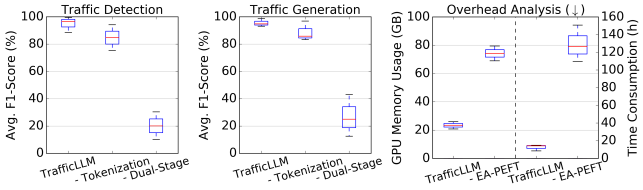
Concept Drift. In Figure 10, we train TrafficLLM on 53-type mobile App traffic and evaluate its generalization performance on no drift dataset, 1-month time interval dataset, and new App version drift dataset. Results indicate that TrafficLLM effectively maintains the detection performance when facing the concept drift scenarios. TrafficLLM outperforms baselines with 4.3%-11.3% and 6.7%-18.6% F1-score on time and version drift respectively. TrafficLLM keeps a strong generalization inherited from LLMs, which captures the common traffic representations in drifted environments to ensure the robustness of detecting the unseen drifted traffic.

APT Attack Detection. We evaluate TrafficLLM on the multi-stage APT attack detection tasks. These attacks usually contain multiple stages of attack targets. For instance, the adversary conducts reconnaissance, establishes footholds, and realizes data exfiltration in a couple of days in the DAPT 2020 dataset. As shown in Figure 11, we train TrafficLLM with the



(a) Robustness of Different LLMs (b) Impact of LLM Parameter Size

Figure 12: Different types and model sizes of LLM adapted in TrafficLLM. The generation performance uses the R-Train and S-Test settings (the same as below).



(a) Traffic Detection (b) Traffic Generation (c) Computation Overhead

Figure 13: The effectiveness of TrafficLLM's core components including tokenization, dual-stage tuning, and EA-PEFT.

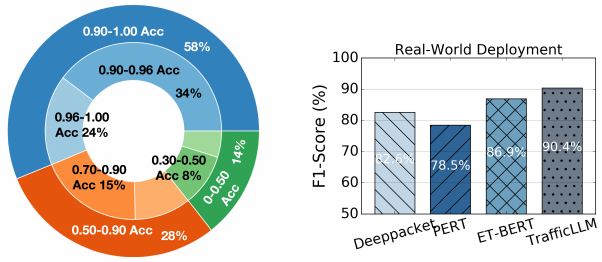
benign and stage-1 APT attack traffic and evaluate whether it can detect the attacks of future stages. Results indicate that TrafficLLM achieves 89.3% F1-score to detect future APT attack traffic. The pattern mining of TrafficLLM can learn robust representation to differ malicious and benign traffic on different stages. The performance improves by 5.7%-26.2% on average compared to the three baselines, which demonstrates TrafficLLM's generalization on unseen attack traffic.

5.4 In-Depth Analysis

Robustness of Different LLMs. To verify whether TraffiLLM is applicable to different LLMs, except for Llama2 and ChatGLM2, we employ additional state-of-the-art LLMs to build TrafficLLM, which includes Vicuna [13], Mistral [34], and Gemma [70]. As shown in Figure 12(a), results indicate that the framework of TraffiLLM can be easily applied to all open-sourced LLMs with strong performance.

Impact of LLM Parameter Size. In Figure 12(b), we explore the impact of LLM's parameter size on traffic detection and generation performance. Although Llama2-13B and ChatGLM2-12B's parameter size is almost 2-fold that of their 7B and 6B versions, the four LLMs achieve similar performance. The 6B model is enough to outperform existing ML-based baselines on traffic detection and generation tasks.

Effectiveness of Core Components. To validate the effectiveness of TrafficLLM's core components, we build three TrafficLLM's variants by removing the traffic-domain tokenizer (- Tokenization), replacing the dual-stage tuning pipeline with default tuning (- Dual-Stage), and replacing EA-PEFT scheme with full fine-tuning (- EA-PEFT). As shown in Figure 13, modifying these components will entail 7.2%-78.7% performance reduction and 927.9% time and 216.2% GPU memory overhead increase among five LLMs mentioned above, which indicates the significance of these components.



(a) Human Evaluation on ATEC 2023 (b) Real-World Deployment

Figure 14: Players' model performance using TrafficLLM on ATEC 2023 competition and the traffic detection performance of EAC tasks under TrafficLLM's real-world deployment.

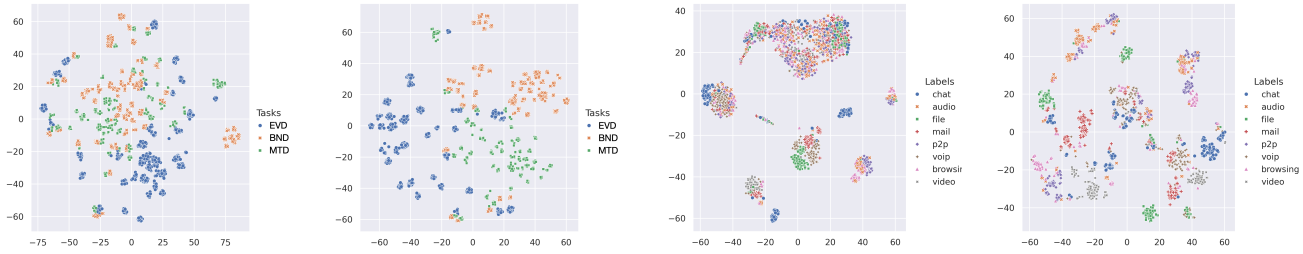
Overhead Analysis. We investigate TrafficLLM's computation overhead on a NVIDA A800-80GB GPU. Training a 6B model like ChatGLM2-6B requires 23GB GPU memory and 14h training time for a new PEFT model update (involving 20,000 training steps on 50,000 task-specific samples). During the inference stage, loading TrafficLLM requires 13GB GPU memory and takes about 0.2s or 10s to generate a predicted label or a 1000-token simulated packet. To reduce the overhead, we consider employing a smaller LLM or using compression methods [84] can help speed up the adaptation.

5.5 Real-World Evaluation

Human Evaluation. To investigate TrafficLLM's effectiveness in research and collect feedback from the community, we integrate TrafficLLM as a race track on ATEC 2023[1], a national LLM competition with 1,901 teams and over 3,000 players from about 200 institutions from November 2023 to March 2024. In this track, players are required to use TrafficLLM's framework with custom instructions and traffic features to tackle MTD, BND, and EVD tasks. As shown in Figure 14(a), 58% of players achieve 90% above accuracy in this competition. Even 24% of player models performed better than 96% Accuracy. After extensive verification by players during the competition, TrafficLLM was proven to be able to easily adapt LLMs to traffic domains with powerful performance.

Real-World Deployment. We leverage the testbed on a Super GPU server (super-SYS-420GP-TNR) and adapt ChatGLM2 to build TrafficLLM's prototype. We use Python Flask to develop TrafficLLM's dialog API. In the real-world setting, we collect 20 types of App traffic on a Huawei WiFi AX3 router. Each type of traffic is generated by operating the App on our Android mobile phone and connecting it to the router. As shown in Figure 14(b), TrafficLLM outperforms baselines by reaching 90.4% F1-score in the real-world EAC task, which indicates TrafficLLM's generalization to real-world scenarios.

[1] ATEC 2023 Website: https://www.atecup.cn/matchHome/100001



(a) LLM Instruction Embeddings (b) TrafficLLM Instruction Embeddings (c) LLM Traffic Embeddings (d) TrafficLLM Traffic Embeddings

Figure 15: The hidden state visualization of task instructions and traffic data in ChatGLM2 and TrafficLLM. TrafficLLM can learn better representations under traffic detection instructions of EVD, BND, and MTD tasks and ISCX Tor 2016 traffic datasets.

Table 8: The task understanding abilities of TrafficLLM and the native LLM based on instructions of downstream tasks.

Model	Task	PR	RC	F1	Acc
Native LLM (Llama2-7B)	Traffic Detection	0.4422	0.6650	0.5312	0.6650
	Traffic Generation	0.5776	0.7600	0.6564	0.7600
TrafficLLM	Traffic Detection	0.9910	0.9925	0.9915	0.9925
	Traffic Generation	0.9935	0.9960	0.9940	0.9960

5.6 Representation Learning

Hidden State Analysis. To explain TrafficLLM's ability to learn text and traffic data, in Figure 15, we show the visualization of TrafficLLM's hidden state representation using T-SNE. Compared to the native LLM, TrafficLLM can learn more distinguishable representations of different traffic analysis instructions through instruction tuning. Moreover, TrafficLLM also learns better traffic embedding for each class due to the task-specific traffic tuning with the customized traffic tokens. TrafficLLM's traffic representations of each type keep clearer boundaries in the feature space, which ensures accuracy across different traffic detection and generation tasks.

Task Understanding. The representation learning ability helps TrafficLLM correctly understand different task instructions from security practitioners. In Table 8, we compare TrafficLLM's performance to the native Llama2-7B, which is required to choose the correct task labels based on the instructions and given options across different detection and generation tasks. Results indicate that TrafficLLM has effectively acquired the domain knowledge for traffic analysis, achieving strong task understanding performance (0.9920 average F1-score) to conduct different downstream tasks.

6 Related Work

Encrypted Traffic Classification. Encrypted traffic classification is a crucial technique for network management and security monitoring. With the inability to inspect the content of encrypted packets directly, researchers have turned to various machine-learning algorithms [6, 23, 51, 64, 69, 75] to analyze patterns, timings, packet sizes, and other metadata to classify traffic. For instance, Van et al. [75] used semi-supervised methods to model device, certificate, and statistical features for mobile app fingerprinting. Taylor et al. [69] utilized packet size to train traffic classifiers with Random Forest. Fu et al. [23] exploited flow interaction graphs to distinguish malicious behaviors from benign traffic. Recent research [29, 40] focused more on model generalization using pre-trained models. For instance, Lin et al. [40] leveraged BERT to build the pre-trained model for multi-type traffic classification tasks. However, existing works only focus on handling traffic data. Unlike these works, TrafficLLM jointly learns security task instructions and traffic features, making TrafficLLM more powerful in various traffic analysis tasks.

Traffic Data Augmentation. Data augmentation has been widely applied in the few-shot scenarios of traffic analysis to increase the amount and diversity of traffic data. For example, Qing et al. [57] utilized distributions of traffic data in the feature space to augment training data for model training. Jan et al. [31] generated labeled datasets for botnet detection using generative models. A bunch of prior works [12, 77, 86] leveraged Generative Adversarial Nets (GANs) to synthesize metadata in the traffic. Unlike these works, TrafficLLM can generate entire packets including accurate headers and synthetic payloads based on traffic sequence modeling.

7 Conclusion

In this paper, we develop a powerful framework to adapt LLMs for network traffic analysis with strong generalization. TrafficLLM employs three core techniques including traffic-domain tokenization to process instructions and traffic data, the dual-stage tuning pipeline to understand text semantics and learn traffic patterns, and the EA-PEFT technique to update model parameters for new scenario adaptation. We evaluate TrafficLLM on 10 open-sourced datasets. Extensive experiments indicate that TrafficLLM shows remarkable generalization abilities across different traffic detection and generation tasks. We release the source code and datasets to facilitate future research and hope that TrafficLLM can serve as the stepping stone for more LLM adaptation designs in the traffic analysis community.

References

[1] Baichuan AI. Baichuan. https://www.baichuan-ai.com/, 2024.

[2] Khaled Al-Naami, Swarup Chandra, Ahmad Mustafa, Latifur Khan, Zhiqiang Lin, Kevin Hamlen, and Bhavani Thuraisingham. Adaptive encrypted traffic fingerprinting with bi-directional dependence. In *ACSAC*, pages 177–188. ACM, 2016.

[3] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. In *NeurIPS*, pages 23716–23736, 2022.

[4] Anthropic. Claude. https://claude.ai/, 2024.

[5] ArcSight. Enterprise security manager. https://www.microfocus.com/en-us/cyberres/secops/arcsightesm, 2023.

[6] Alireza Bahramali, Ramin Soltani, Amir Houmansadr, Dennis Goeckel, and Don Towsley. Practical traffic analysis attacks on secure messaging applications. In *NDSS*. The Internet Society, 2020.

[7] Elaheh Biglar Beigi, Hossein Hadian Jazi, Natalia Stakhanova, and Ali A Ghorbani. Towards effective feature selection in machine learning-based botnet detection approaches. In *CNS*, pages 247–255. IEEE, 2014.

[8] Giampaolo Bovenzi, Davide Di Monda, Antonio Montieri, Valerio Persico, and Antonio Pescapé. Few shot learning approaches for classifying rare mobile-app encrypted traffic samples. In *INFOCOM Workshops*, pages 1–6. IEEE, 2023.

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *NeurIPS*, 2020.

[10] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Independent comparison of popular dpi tools for traffic classification. *Computer Networks*, 76:75–89, 2015.

[11] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramèr, and Chiyuan Zhang. Quantifying memorization across neural language models. In *ICLR*, 2023.

[12] Adriel Cheng. Pac-gan: Packet generation of network traffic using generative adversarial networks. In *IEMCON*, pages 0728–0734. IEEE, 2019.

[13] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[14] Cisco. Cisco span. https://www.cisco.com/c/en/us/support/docs/swit-ches/catalyst-6500-series-switches/10570-41.html, 2023.

[15] Jiaxi Cui, Zongjian Li, Yang Yan, Bohua Chen, and Li Yuan. Chatlaw: Open-source legal large language model with integrated external knowledge bases. *arXiv preprint arXiv:2306.16092*, 2023.

[16] CSIC 2010 Dataset. Csic. http://www.isi.csic.es/dataset/, 2024.

[17] Wladimir De la Cadena, Asya Mitseva, Jens Hiller, Jan Pennekamp, Sebastian Reuter, Julian Filter, Thomas Engel, Klaus Wehrle, and Andriy Panchenko. Trafficsliver: Fighting website fingerprinting attacks with traffic splitting. In *CCS*, pages 1971–1985. ACM, 2020.

[18] Xinhao Deng, Qilei Yin, Zhuotao Liu, Xiyuan Zhao, Qi Li, Mingwei Xu, Ke Xu, and Jianping Wu. Robust multi-tab website fingerprinting attacks in the wild. In *SP*, pages 1005–1022. IEEE, 2023.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186. Association for Computational Linguistics, 2019.

[20] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023.

[21] Wireshark Foundation. Wireshark - go deep. https://www.wireshark.org/, 2024.

[22] Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. Realtime robust malicious traffic detection via frequency domain analysis. In *CCS*, pages 3431–3446. ACM, 2021.

[23] Chuanpu Fu, Qi Li, and Ke Xu. Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis. In *NDSS*. The Internet Society, 2023.

[24] Zhuoqun Fu, Mingxuan Liu, Yue Qin, Jia Zhang, Yuan Zou, Qilei Yin, Qi Li, and Haixin Duan. Encrypted malware traffic detection via graph-based network analysis. In *RAID*, pages 495–509. USENIX Association, 2022.

[25] Gerard Drapper Gil, Arash Habibi Lashkari, Mohammad Mamun, and Ali A Ghorbani. Characterization of encrypted and vpn traffic using time-related features. In *ICISSP*, pages 407–414. SciTePress, 2016.

[26] Google. Gemini - google deepmind. https://deepmind.google/technologies/gemini/, 2024.

[27] Satyandra Guthula, Navya Battula, Roman Beltiukov, Wenbo Guo, and Arpit Gupta. netfound: Foundation model for network security. *arXiv preprint arXiv:2310.17025*, 2023.

[28] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security*, pages 1187–1203. USENIX Association, 2016.

[29] Hong Ye He, Zhi Guo Yang, and Xiang Ning Chen. Pert: Payload encoding representation from transformer for encrypted traffic classification. In *ITU K*, pages 1–8. IEEE, 2020.

[30] Sihao Hu, Tiansheng Huang, Ka-Ho Chow, Wenqi Wei, Yanzhao Wu, and Ling Liu. Zipzap: Efficient training of language models for large-scale fraud detection on blockchain. In *WWW*, pages 2807–2816, 2024.

[31] Steve TK Jan, Qingying Hao, Tianrui Hu, Jiameng Pu, Sonal Oswal, Gang Wang, and Bimal Viswanath. Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In *SP*, pages 1190–1206. IEEE, 2020.

[32] Mobin Javed and Vern Paxson. Detecting stealthy, distributed SSH brute-forcing. In *CCS*, pages 85–96. ACM, 2013.

[33] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Survey*, 55(12):248:1–248:38, 2023.

[34] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[35] Minghao Jiang, Mingxin Cui, Chang Liu, Gaopeng Gou, Gang Xiong, and Zhen Li. Zero-relabelling mobile-app identification over drifted encrypted network traffic. *Computer Networks*, 228:109728, 2023.

[36] Ashraf M Kibriya, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes. Multinomial naive bayes for text categorization revisited. In *AI 2004*, pages 488–499. Springer, 2005.

[37] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of tor traffic using time based features. In *ICISSP*, volume 2, pages 253–262. SciTePress, 2017.

[38] Peiyang Li, Ye Wang, Qi Li, Zhuotao Liu, Ke Xu, Ju Ren, Zhiying Liu, and Ruilin Lin. Learning from limited heterogeneous training data: Meta-learning for unsupervised zero-day web attack detection across web domains. In *CCS*, pages 1020–1034, 2023.

[39] Kunda Lin, Xiaolong Xu, and Honghao Gao. Tscrnn: A novel classification scheme of encrypted traffic based on flow spatiotemporal features for efficient management of iiot. *Computer Networks*, 190:107974, 2021.

[40] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In *WWW*, pages 633–642. ACM, 2022.

[41] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. Fs-net: A flow sequence network for encrypted traffic classification. In *INFOCOM*, pages 1171–1179. IEEE, 2019.

[42] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

[43] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021.

[44] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3):1999–2012, 2020.

[45] Xuying Meng, Chungang Lin, Yequan Wang, and Yujun Zhang. Netgpt: Generative pretrained transformer for network traffic. *arXiv preprint arXiv:2304.09513*, 2023.

[46] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. Interpreting deep learning-based networking systems. In *SIGCOMM*, pages 154–171. ACM, 2020.

[47] Meta. Meta llama 3. https://llama.meta.com/llama3/, 2024.

[48] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. In *NDSS*. The Internet Society, 2018.

[49] MohammadrezaMontazeriShatoori, Logan Davidson, Gurdip Kaur, and Arash Habibi Lashkari. Detection of doh tunnels using time-series classification of encrypted traffic. In *CyberSciTech*, pages 63–70. IEEE, 2020.

[50] Sowmya Myneni, Ankur Chowdhary, Abdulhakim Sabur, Sailik Sengupta, Garima Agrawal, Dijiang Huang, and Myong Kang. Dapt 2020-constructing a benchmark dataset for advanced persistent threats. In *DMLSD*, pages 138–163. Springer, 2020.

[51] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *CCS*, pages 2053–2069. ACM, 2017.

[52] Sanghak Oh, Minwook Lee, Hyunwoo Lee, Elisa Bertino, and Hyoungshick Kim. AppSniffer: Towards robust mobile app fingerprinting against vpn. In *WWW*, pages 2318–2328. ACM, 2023.

[53] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[54] OpenAI. Chatgpt. https://chat.openai.com/, 2024.

[55] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *NDSS*. The Internet Society, 2016.

[56] Giancarlo Pellegrino, Martin Johns, Simon Koch, Michael Backes, and Christian Rossow. Deemon: Detecting csrf with dynamic analysis and property graphs. In *CCS*, pages 1757–1771. ACM, 2017.

[57] Yuqi Qing, Qilei Yin, Xinhao Deng, Yihao Chen, Zhuotao Liu, Kun Sun, Ke Xu, Jia Zhang, and Qi Li. Low-quality training data only? a robust framework for detecting encrypted malicious network traffic. In *SP*. IEEE, 2023.

[58] Rapid7. Insightidr. https://www.rapid7.com/products/insightidr/, 2024.

[59] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. In *NDSS*. The Internet Society, 2018.

[60] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? In *NeurIPS*, 2023.

[61] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*. Association for Computational Linguistics, 2016.

[62] Meng Shen, Jinpeng Zhang, Liehuang Zhu, Ke Xu, and Xiaojiang Du. Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. *IEEE Transactions on Information Forensics and Security*, 16:2367–2380, 2021.

[63] Hongtao Shi, Hongping Li, Dan Zhang, Chaqiu Cheng, and Xuanxuan Cao. An efficient feature generation approach based on deep learning and feature selection techniques for traffic classification. *Computer Networks*, 132:81–98, 2018.

[64] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. Encrypted dns–> privacy? a traffic analysis perspective. In *NDSS*. The Internet Society, 2019.

[65] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *CCS*, pages 1928–1943. ACM, 2018.

[66] Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, and Denny Zhou. Fast wordpiece tokenization. In *EMNLP*, pages 2089–2103. Association for Computational Linguistics, 2021.

[67] CISCO Systems. Cisco secure network analytics. https://www.cisco.com/site/us/en/products/security/security-analytics/secure-network-analytics/index.html, 2024.

[68] Ruming Tang, Zheng Yang, Zeyan Li, Weibin Meng, Haixin Wang, Qi Li, Yongqian Sun, Dan Pei, Tao Wei, Yanfei Xu, et al. Zerowall: Detecting zero-day web attacks through encoder-decoder recurrent neural networks. In *INFOCOM*, pages 2479–2488. IEEE, 2020.

[69] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security*, 13(1):63–78, 2017.

[70] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[71] Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. Large language models in medicine. *Nature medicine*, 29(8):1930–1940, 2023.

[72] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, Daniel Margolis, Vern Paxson, and Elie Bursztein. Data breaches, phishing, or malware?: Understanding the risks of stolen credentials. In *CCS*, pages 1421–1434. ACM, 2017.

[73] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[74] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[75] Thijs Van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J Dubois, Martina Lindorfer, David Choffnes, Maarten Van Steen, and Andreas Peter. Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *NDSS*, volume 27. The Internet Society, 2020.

[76] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[77] Pan Wang, Shuhang Li, Feng Ye, Zixuan Wang, and Moxuan Zhang. Packetcgan: Exploratory study of class imbalance for encrypted traffic classification using cgan. In *ICC*, pages 1–7. IEEE, 2020.

[78] Qineng Wang, Chen Qian, Xiaochang Li, Ziyu Yao, and Huajie Shao. Lens: A foundation model for network traffic in cybersecurity. *arXiv preprint arXiv:2402.03646*, 2024.

[79] Tao Wang. High precision open-world website fingerprinting. In *SP*, pages 152–167. IEEE, 2020.

[80] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. Malware traffic classification using convolutional neural network for representation learning. In *ICOIN*, pages 712–717. IEEE, 2017.

[81] Yue Wang, Hung Le, Akhilesh Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. Codet5+: Open code large language models for code understanding and generation. In *EMNLP*, pages 1069–1088, 2023.

[82] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Transaction of Machine Learning Research*, 2022, 2022.

[83] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

[84] Canwen Xu and Julian McAuley. A survey on model compression and acceleration for pretrained language models. In *AAAI*, volume 37, pages 10566–10575, 2023.

[85] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. {CADE}: Detecting and explaining concept drift samples for security applications. In *USENIX Security*, pages 2327–2344, 2021.

[86] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. Practical gan-based synthetic ip header trace generation using netshare. In *SIGCOMM*, pages 458–472. ACM, 2022.

[87] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. GLM-130B: an open bilingual pre-trained model. In *ICLR*, 2023.

[88] Duzhen Zhang, Yahan Yu, Jiahua Dong, Chenxing Li, Dan Su, Chenhui Chu, and Dong Yu. Mm-llms: Recent advances in multimodal large language models. In *ACL*, pages 12401–12430. Association for Computational Linguistics, 2024.

[89] Haozhen Zhang, Le Yu, Xi Xiao, Qing Li, Francesco Mercaldo, Xiapu Luo, and Qixu Liu. TFE-GNN: A temporal fusion encoder using graph neural networks for fine-grained encrypted traffic classification. In *WWW*, pages 2066–2075. ACM, 2023.

[90] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[91] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. Parallelized stochastic gradient descent. In *NeurIPS*, 2010.

Table 9: Examples of Gemini (Gemini 1.5), Claude (Claude 3 Sonnet), and ChatGPT (GPT-4)'s responses to the instructions for conducting traffic detection and traffic generation tasks.

Task	Instruction	Response		
Traffic Detection	❶ Please conduct the ENCRYPTED APP CLASSIFICATION TASK to determine which APP category the encrypted traffic belongs to. The categories include 'Adobe, Alipay, ..., and Booking'. <packet>: 04d9... [Encrypted App Classification]	**Gemini:** Based on the provided traffic data, the predicted app category is: Alipay. Explanation: The packet size and structure may resemble previously observed patterns associated with Alipay's protocols ... [Random Guess]	**Claude:** The most likely app category for the provided encrypted traffic is Apple. The packet data contains patterns and characteristics that strongly suggest the traffic is related to Apple services ... [Random Guess]	**ChatGPT:** To determine which class encrypted traffic belongs to based on packet data, a detailed analysis of patterns and features within the encrypted traffic itself is required ... [Only Recommendations]
	❷ The following is a segment of network traffic data, you need to analyze whether it is generated by the communication behavior of malware. If so, please categorize it as Cridex, Geodo, ..., and Zeus. <packet>: 04... [Malware Traffic Detection]	**Gemini:** The provided packet snippet appears to be a DNS query, but without additional context and analysis, it's difficult to definitively determine if it's generated by malware. However, we can examine ... [Only Description]	**Claude:** Without additional context or analysis, it's difficult to definitively determine if this specific packet is generated by malware communication behavior. However, I can provide an overview ... [Only Description]	**ChatGPT:** Considering the DNS query characteristics, and without additional payload decryption or further traffic analysis, this packet could be related to families like Zeus or Shifu ... [Random Guess]
Traffic Generation	❶ Based on the protocol of fields, traffic features, and payloads of different types of traffic in your knowledge. Please generate a packet of YouTube with the pcap format. [Youtube Traffic Generation]	**Gemini:** Due to the limitations of my knowledge cutoff and the dynamic nature of YouTube traffic, I cannot directly generate a specific packet in hexadecimal format ... [Only Recommendations]	**Claude:** Unfortunately, I cannot generate packet capture (pcap) files or raw network traffic data. As an AI assistant without direct access to network devices or the ... [Only Recommendations]	**ChatGPT:** IP Header: Source IP: 192.168.1.2, Destination IP: 216.58.214.14, Protocol: TCP, TCP Header: Source Port: 49152, Destination Port: 443, Flags: ... [Random Generation]
	❷ I need a packet of Weibo traffic. Based on Weibo's traffic feature, please provide me with a packet of Weibo's traffic data with the pcap format. [Weibo Traffic Generation]	**Gemini:** Unfortunately, due to limitations in my capabilities and knowledge cutoff, I cannot directly generate a pcap file containing Weibo traffic. However, ... [Only Recommendations]	**Claude:** I'm afraid I cannot provide you with an actual packet capture (pcap) file containing Weibo traffic data. As an AI assistant without direct access ... [Only Recommendations]	**ChatGPT:** To obtain a real packet of Weibo's traffic in PCAP format, you would typically use a network traffic capturing tool like Wireshark ... [Only Recommendations]

Appendix

A Open Science & Ethical Considerations

A.1 Open Science

To facilitate future research to adapt LLMs in traffic analysis domain, we release the source code of TrafficLLM and the datasets at https://github.com/ZGC-LLM-Safety/TrafficLLM. The dataset is consists of over 0.4M tuning data that include human instructions and traffic features supervised by manual annotation. To the best of our knowledge, this is the largest LLM adaptation dataset for traffic domain to date. To enhance the reproducibility of our work, we mainly use open-sourced datasets and LLMs to conduct the experiments in this paper. All the original traffic data and LLMs are publicly available in their released repositories.

A.2 Ethical Considerations

In this paper, we discuss ethical considerations about dataset construction, human evaluation, and real-world deployment.
Dataset Construction. We leverage the human instruction and traffic data to construct the tuning data for LLM adaptation. The released traffic data are all extracted from the open-sourced datasets. For the human instruction dataset, we invite five network security practitioners and students to generate task-specific instructions with manual annotation and AI assistance. The instructions are required not including corporate confidentiality and privacy data. We have submitted the data review material to our institutional ethics review body (IRB). Our work has been approved by IRB to ensure ethical soundness and justification.
Human Evaluation. We evaluate TrafficLLM's effectiveness with extensive participation of players in the ATEC 2023 competition. All the players have signed an informed consent agreement to allow us to collect the competition data. In the competition system, each player is assigned a model submitter ID. We did not over-explore data involving personal information and designed a detailed exit mechanism to remove user records only necessary statistical data used in our experiments. The released statistical data has been approved by the competition organizer and IRB.
Real-World Deployment. We deploy TrafficLLM on a real-world GPU server and release the API to verify the traffic mirror on a Huawei router for EAC tasks. The traffic data is collected by operating our Android mobile phone with 100 Apps. We further guarantee that our collection process does not disrupt or harm other hosts or targets.

B Details of LLM Investigation

We investigate the capabilities of existing LLMs and earlier PLMs to achieve the traffic analysis tasks in Section 2.2. For a more detailed study of existing LLMs, Table 9 shows examples of 3 state-of-the-art LLMs' responses to traffic analysis instructions. We can see that existing LLMs tend to give random guesses, only descriptions of the traffic data, or only recommendations of manual steps when receiving 3 types of traffic analysis instructions. For instance, when facing a traffic detection instruction of EAC or MTD tasks, native LLM usually generate a random label based on its knowledge (e.g., Alipay), give a detailed description of the traffic features (e.g., IP and protocol), and give a recommendation to use Wireshark or other network traffic analyzer. This investigation demon-

Table 10: The traffic features extracted from the datasets for traffic detection except for WF and WAD tasks.

ID	Layer	Feature	Description
1	IP	ip.version	Internet Protocol version. IPv4 is 4. IPv6 is 6.
2		ip.hdr_len	IP Header Length field in IPv4 headers.
3		ip.dsfield	Differentiated Services field in IPv4 headers.
4		ip.len	Total length of the IP packet.
5		ip.id	Identification field to identify packet fragmentation.
6		ip.flags	IP Flags that control and manage IP fragmentation.
7		ip.frag_offset	Fragment location in the datagram.
8		ip.ttl	Time To Live field that manages the lifetime of the packet.
9		ip.proto	Protocol of the payload in the IP packet.
10		ip.checksum	IP Checksum that ensure the integrity of the header.
11		ip.src	Source IP address in packets.
12		ip.dst	Destination IP address in packets.
13	TCP	tcp.srcport	Source port number in TCP headers.
14		tcp.dstport	Destination port number in TCP headers.
15		tcp.stream	TCP flow number of the packet.
16		tcp.len	Length of the TCP payload.
17		tcp.seq	Sequence number of TCP packets.
18		tcp.nxtseq	Next sequence number of TCP packets.
19		tcp.ack	Acknowledgment number for TCP transmission.
20		tcp.hdr_len	TCP Header Length field in TCP headers
21		tcp.flags	TCP Flags that control and manage TCP connections.
22		tcp.window_size	Window size for flow control in TCP connections.
23		tcp.checksum	TCP Checksum that ensure the integrity of the header.
24		tcp.urgent_pointer	Urgent data identifier in TCP headers.
25		tcp.time_relative	Amount of time since the first packet captured.
26		tcp.time_delta	Time difference between two packets in a TCP session.
27		tcp.bytes_in_flight	Total amount of unacknowledged data.
28		tcp.push_bytes_sent	Number of bytes sent in TCP segments.
29		tcp.segment	TCP segment flag.
30		tcp.segment.count	Number of TCP segments that have been transmitted.
31		tcp.reassembled.length	Total amount of data that has been reassembled.
32	UDP	udp.srcport	Source port number in UDP headers.
33		udp.dstport	Destination port number in UDP headers.
34		udp.length	Length of the entire UDP packet.
35		udp.checksum	UDP Checksum that ensure the integrity of the header.
36		udp.stream	UDP flow number of the packet.
37		udp.data.len	Length of the UDP payload.
38	Payload	payload	Hexadecimal raw payload in the packet.

strates that existing LLMs still cannot obtain accurate results for traffic detection and generation, which indicates the strong motivation to build TrafficLLM.

C Details of Traffic Features

Based on the experience of previous work [38, 65, 75], we inherit these feature extraction outcomes to encode original traffic data as the features in TrafficLLM. For WF and WAD tasks, TrafficLLM extract packet direction sequences and HTTP request headers as the tuning data. For the other traffic detection tasks, we extract traffic fields and statistical features in Table 10 for traffic pattern learning in LLMs. For traffic generation tasks, we extract 5-tuples information and hexadecimal raw data for traffic simulation.

D Details of Experimental Settings

D.1 Instruction Dataset

The human instructions for LLM adaptation are constructed by 5 experts in the network security industries and PH.D. students majoring in cybersecurity. They are required to generate instruction templates using domain knowledge across different traffic detection and generation tasks and extend the scale via generative AI assistants. We generate expert instructions according to the backgrounds of 10 traffic classification tasks and 229 types of traffic generation tasks. We collect diverse AI-generated instructions using ChatGPT [54] with a rewrite instruction like 'The following is a traffic analysis instruction provided by network security experts. Please consider different scenarios, security goals, question subjects, writing styles, and text descriptions. Rewrite the following instructions and generate 20 new different traffic analysis instructions'. We show instruction examples of TBD, MDD, and BND tasks as follows.

Instruction Examples across Different Tasks

TBD Task Instruction$_1$: The Tor network transmits traffic through multiple layers of encryption and relay nodes, providing users with online privacy and anonymity. Please classify the traffic application or behavior according to the traffic data I provided.
TBD Task Instruction$_2$: Hello, I want to monitor whether there is any dark web access in the network. This is a piece of traffic data collected from it. Can you help me identify whether there is any behavior using the Tor network? If so, what behavior or specific application is it under the Tor network?
MDD Task Instruction$_1$: The following traffic may be DoH data generated by an encrypted DNS server. Please help me determine whether it contains malicious DoH behavior.
MDD Task Instruction$_2$: You are a reliable security model that can effectively analyze the specific categories of traffic. Now I provide the following DoH traffic. Please identify the category label. The specific categories include malicious and benign.
BND Task Instruction$_1$: The traffic data generated by the communication between the control server and the controlled host is obviously different from that of the normal network. The data is as follows. Please guess the network type to which this data belongs.
BND Task Instruction$_2$: This is a piece of network traffic data, which may contain the behavior of a network host infected by a bot program. I need to confirm the specific category of this traffic.

D.2 Overview of Baselines

In this paper, we compare the performance of TrafficLLM with 15 baselines across different tasks, including state-of-the-art traffic detection and generation algorithms.
Traffic Detection. The following provides an overview of the traffic classification baseline used in our evaluation.

- **AppScanner** [69]. AppScanner is a statistical feature method using 54 statistical features for smartphone App identification based on ML models.

- **CUMUL** [55]. CUMUL extracts packet size, direction, and ordering features for website fingerprinting on the Tor network.
- **BIND** [2]. BIND uses bi-directional burst features for encrypted traffic fingerprinting in a wide range.
- **k-fingerprinting (K-FP)** [28]. K-FP extracts the most important features using random forests for large-scale traffic classification.
- **FlowPrint** [75]. FlowPrint extracts statistical features for encrypted traffic classification using the semi-supervised clustering method.
- **FS-Net** [41] FS-Net uses RNN models to learn packet length sequences for traffic classification.
- **Deep Fingerprinting (DF)** [65]. DF uses deep Convolutional Neural Networks to realize website fingerprinting.
- **GraphDApp** [62]. GraphDApp extracts traffic interaction graphs and uses Graph Neural Networks to detect traffic.
- **TSCRNN** [39]. TSCRNN combines CNN and RNN to extract abstract features of the flow for traffic classification.
- **Deeppacket** [44]. Deeppacket uses deep Convolutional Neural Networks to realize encrypted traffic classification.
- **PERT** [29]. PERT uses a Transformer-based encoder and pre-training technique to learn traffic representation.
- **ET-BERT** [40]. ET-BERT pre-trains BERT with large-scale traffic data to realize traffic classification across different tasks.

Note that we choose these methods to cover a wide range of traffic detection tasks. For WF and WAD tasks, we do not evaluate statistical feature methods and GraphDApp since they can not adapt to the dataset format to extract effective features. We modify the feature extraction methods to scale the deep learning-based methods on WF and WAD datasets.

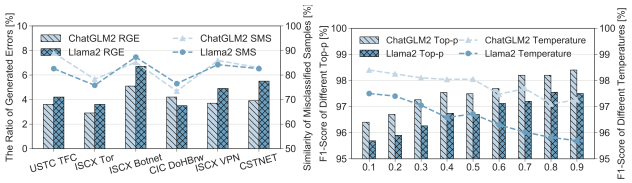
Traffic Generation. We use the following traffic generation algorithms as the baselines in the experiments.

- **Netshare** [86]. Netshare uses the GAN-based method to generate IP header traces at large scale.
- **PacketCGAN** [12]. PacketCGAN uses conditional GANs to generate the encrypted traffic using bit vectors.
- **PAC-GAN** [77]. PAC-GAN uses CNN GAN to encode network packet into images and use them to generate packets.

D.3 Real-World Evaluation Setup

Human Evaluation Setup. To investigate the effectiveness of deploying TrafficLLM in the community, we conduct the human evaluation of TrafficLLM by integrating the framework as a track in the ATEC 2023 competition. Table 11 shows the detail of the competition dataset to evaluate TrafficLLM. As one of the organizers of ATEC 2023, we join the development of the competition platform. The platform can run the model image submitted by players and record the score on a real-time ranking web page. We analyze the results to collect TrafficLLM's performance under the large research group.

Real-World Deployment Setup. In the real-world deployment scenario, we implement TrafficLLM prototype and its API on a super GPU server. We capture the real-world traffic data on a Huawei WiFi AX3 router, which is also shown in Table 11. The traffic data is treated as the traffic mirror and is submitted to TrafficLLM with EAC task instructions through the API. Experiments indicate that TrafficLLM can reach 90.4 %F1-score. In this setting, TrafficLLM API takes an average of 0.3s to return to the predicted labels.



(a) RGE and SMS Metrics (b) Top-p and Temperature

Figure 16: Left: The ratio of generated errors (RGE) and the similarity of misclassified samples (SMS) metric; Right: The performance of different top-p and temperature in TrafficLLM.

Table 11: The details of traffic data in the human evaluation on ATEC 2023 and real-world deployment experiments.

Experiment	Task	#Flows	#Packet	#Labels
Human Evaluation on ATEC 2023	MTD	2,855	9,458	20
	BND	16,930	52,278	5
	EVD	1,025	3,392	19
Real-World Deployment	EAC	2,106	8,170	20

E Hallucination Evaluation

LLM is prone to hallucination [33] due to the token generation with word sampling strategies. This may influence the detection accuracy during the inference. In the hallucination evaluation experiment, we collect the prediction results from 10 rounds of inference using the same test set. We define the different responses for the same test sample input as the generated errors. Figure 16(a) shows the ratio of generated errors in the misclassified samples and the average Jaccard similarity between the misclassified samples and the dataset of the misclassified label. Results indicate that 3.9% and 4.7% of the misclassified output are generated errors on average when using ChatGLM2 and Llama2 as the foundation model in TrafficLLM. These misclassified samples usually keep a high similarity (i.e., 82.4% and 81.5% on average) to the datasets of misclassified labels, which we consider as the reason for raising the hallucination issues. To address the problem, Figure 16(b) measures the performance of different Top-p and Temperature parameters setting in the sampling strategy. A higher Top-p and a lower Temperature parameter can help the model keep strong confidence when predicting labels and mitigate the hallucination in traffic detection.